ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (968-972), Month: April - June 2015, Available at: www.researchpublish.com

An Efficient Prediction of Range Aggregation Based On Dagger

[1] Anitha.J, [2]Maheswari.D, [3]MadanMohan.M

[1,2] PG Scholar, [3]Assistant Professor, Department of Computer Science and Engineering,
Ranganathan Engineering College, Coimbatore, India

***Abstract*: An efficient prediction of range aggregation based on dagger uses bundled range aggregation, which can be regarded as the simultaneous execution of a range aggregate query on multiple datasets, returning a result for each dataset. Bundled range aggregation, which is conceptually equivalent to running a range aggregate query separately on multiple datasets, returning the query result on each dataset. In particular, the queried datasets can be arbitrarily chosen from a large number (hundreds or even thousands) of candidate datasets. The challenge is to minimize the query cost no matter how many and which datasets are selected. We propose DAGGER (Dataset Aggregation), an iterative algorithm that trains a deterministic policy that achieves good performance guarantees under its induced distribution of states. we found that Dagger is more stable and learns faster, while being more robust with respect to the choice of learning rates and action costing. These advantages are more pronounced in the parameter-free versions of the algorithms which avoid stochastic cost estimates and need simpler expert policy definitions. Finally, we assessed the effect of the learning rate in complex structured prediction tasks in which mistaken predictions can inhibit imitation learning algorithms from exploring useful parts of the training data.**

***Keywords:* Aggregation, range search, index, dagger.**

I. INTRODUCTION

Range aggregation is a multi-bucket value source based aggregation that enables the user to define a set of ranges - each representing a bucket. During the aggregation process, the values extracted from each document will be checked against each bucket range and "bucket" the relevant/matching document. Note that this aggregation includes the from value and excludes the to value for each range. Range aggregation computes an aggregate result about the data items satisfying a range predicate.

To be specific, denote by D a dataset where each item has a key in the real domain. Given an interval r, let D(r) be the set of items in D whose keys are covered by r.

A range count query returns the number of items in D(r). Sometimes each item may carry a real-valued weight. In this case, a range sum query returns the total weight of the items in D(r). Similarly, range aggregation can also be performed using other aggregate functions. For example, a range average query reports the average weight of the items in D(r).Bundled range aggregation is motivated by the fact that, in applications where data are naturally divided into categories, a user is often interested in some, but not all, of the categories. For example, consider a crime database that stores, for each city in the US, the number of crimes each day. Here, every city is a category, in which each item is a pair of (date, crime number), where date is the item's search key, and crime number is its weight. The B-tree is a well-known structure for solving the classic problem of range reporting. With slight augmentation, aB-tree can be changed into an aggregate B-tree (aB-tree), which supports range aggregation effectively. There are two straight forward ways to apply aB-trees for bundled range aggregation. One aB-tree on all categories and One aB-tree for each category. We are not aware of any previous study dedicated to bundled range aggregation. This problem, given its importance both as a standalone operator and as the building brick of more complex problems, deserves specialized efforts to improve the

above straightforward methods. By using some methods like preliminaries, the aggregate bundled b-tree, dynamic maintenance and then more. In dynamic maintenance patching and upgrade algorithms are used. The algorithms are algorithm spilt and algorithm merge. The data and queries are used in experiments.

II. RELATED WORK

A framework for supporting OLAP operations over spatiotemporal data is described. We argue that the spatial and temporal dimensions should be modeled as a combined dimension on the data cube and present data structures, which integrate spatiotemporal indexing with pre-aggregation. While the well-known materialization techniques require a-prior knowledge of the grouping hierarchy, we develop methods that utilize the proposed structures for efficient execution of ad-hoc group-bys. Our techniques can be used for both static and dynamic dimensions. Spatio-temporal databases have received considerable attention during the past few years due to the accumulation of large amounts of multi-dimensional data evolving in time, and the emergence of novel applications such as traffic supervision, and mobile communication systems. Research has focused on modeling, indexing and query processing issues for problems involving historical information retrieval, motion and trajectory preservation, future location estimation etc. All these approaches assume that object locations are individually stored, and queries ask for objects that satisfy some spatio-temporal condition. Throughout the paper we assume that the spatial dimension at the finest granularity consists of a set o regions. This paper addresses these problems by proposing several indexing solutions. First we deal with static spatial dimensions and focus on queries that ask for aggregated data in a query window over a continuous time interval. Numerous indexes have been proposed for indexing spatial and temporal databases. In data warehouses and OLAP the most common conceptual model for data ware houses is the multidimensional data view.

Multiversion SB-Tree (MVSB-tree) for incrementally maintaining and efficiently computing the dominance-sum queries and in turn range-temporal aggregate queries. Computing temporal aggregates is an important but costly operation for applications that main-tain time-evolving data. Due to the large volume of such data, performance improvements for temporal aggregate queries are critical. Previous approaches have aggregate predicates that involve only the time dimension. In this paper we examine techniques to compute temporal aggregates that include key-range predicates as well. In particular we concentrate on the SUM aggregate, while count is a special case. To handle arbitrary key ranges, previous methods would need to keep a separate index for every possible key range. We analyze the performance of our approach and present experimental results that show its efficiency. Furthermore, we address a novel and practical variation called functional range-temporal aggregates. Here, the value of any record is a function over time. The meaning of aggregates is altered such that the contribution of a record to the aggregate result is proportional to the size of the intersection between the record's time interval and the query time interval. Both analytical and experimental results show the efficiency of our result. This paper proposes an indexing technique for computing range-temporal aggregates with guaranteed logarithmic access time. First, the plain range-temporal aggregate query is reduced into several sub-queries. Next, a index structure called the Multiversion SB-Tree (MVSB-tree) is proposed to solve these sub-queries. The proposed structure incorporates features from both the SB-Tree and the Multiversion B+-tree (MVBT) This structure supports efficient queries and yet allows similarly efficient updates. The SB-tree was proposed to answer the temporal aggregate queries without range predicates. The MVSB-tree has very fast (logarithmic) query time and update time, at the expense of a small space overhead. MVSB-tree should prove that it is asymptotically optimal if exact answers are required. Our result of this paper examined temporal aggregate queries in the presence of key-range predicates. Such queries allow warehouse managers to focus on tuples grouped by some key range over a given time interval. We considered both plain and functional range-temporal aggregates. These problems are reduced to dominance-sum queries.

Indexing uncertain categorical data two index structures for efficiently searching uncertain categorical data, one based on the R-tree and another based on an inverted index structure. Using these structures, we provide a detailed description of the probabilistic equality queries they support. Experimental results using real and synthetic datasets demonstrate how these index structures can effectively improve the performance of queries through the use of internal probabilistic information. Uncertainty is prevalent in many application domains. Consider for example a data cleaning application that automatically detects and corrects errors. This paper addresses the problem of indexing uncertain categorical data represented as a set of values with associated probabilities. We propose two different index structures. We show that these structures support a broad range of probabilistic queries over uncertain data, including the typical equality, probability threshold, and top-K queries. Our index structures can also be used for queries that are only meaningful for uncertain data

such as distribution similarity queries. The new indexes are shown to provide efficient execution of these queries with good scalability through experimental validation using real and synthetic data. Under the categorical uncertainty model, a relation can have attributes that are allowed to take on uncertain values. This definition of equality is a natural extension of the usual equality operator for certain data. As with the regular equality operator, this uncertain version can be used to define operations such as joins over uncertain attributes. There are two alternative strategies to split an overfull page: top-down and bottom-up. In the top-down strategy, we pick two children MBRs whose boundaries are distribution ally farthest from each other according to the divergence measures. With these two serving as the seeds for two clusters, all other UDAs are inserted into the closer cluster. The real dataset is generated by text clustering/ categorization of customer service constraints for a major cell phone service provider in the context of CRM databases.

III. DAGGER ALGORITHM

DAGGER (Dataset Aggregation), an iterative algorithm that trains a deterministic policy that achieves good performance guarantees under its induced distribution of states. In its simplest form, the algorithm proceeds as follows. At the first iteration, it uses the expert's policy to gather a dataset of trajectories D and train a policy that best mimics the expert on those trajectories. Then at iteration n, it uses n to collect more trajectories and adds those trajectories to the dataset D. The next policy is the policy that best mimics the expert on the whole dataset D. In other words, DAGGER proceeds by collecting a dataset at each iteration under the current policy and trains the next policy under the aggregate of all collected datasets. The intuition behind this algorithm is that over the iterations, we are building up the set of inputs that the learned policy is likely to encounter during its execution based on previous experience (training iterations).This algorithm can be interpreted as a Follow-The-Leader algorithm in that at iteration n we pick the best policy in hindsight, i.e. under all trajectories seen so far over the iterations. The proposed method solves the greater than B page size problem. DAGGER (Dataset Aggregation), an iterative algorithm that trains a deterministic policy that achieves good performance guarantees under its induced distribution of states

- Dataset Collection and Pre-processing
- Classification of attributes
- Dataset Aggregation Process
- Feature selection or dimensionality reduction

3.1 Dataset Collection and Pre-Processing:

A data set is a collection of data, usually presented in tabular form. Each column represents a particular variable. Each row corresponds to a given member of the data set in question. It lists values for each of the variables, such as height and weight of an object or values of random numbers. Each value is known as a datum. The data set may comprise data for one or more members, corresponding to the number of rows. The values may be numbers, such as real numbers or integers, for example representing a person's height in centimeters, but may also be nominal data (i.e., not consisting of numerical values), for example representing a person's ethnicity. More generally, values may be of any of the kinds described as a level of measurement. For each variable, the values will normally all be of the same kind. However, there may also be "missing values", which need to be indicated in some way.

3.2.Classification of Attributes:

Classifiers are generated for each class of event using relevant features for the class and classification algorithm. Binary classifiers are derived from the training sample by considering all classes other than the current class as other, e.g., C normal will consider two classes: normal and other. The purpose of this phase is to select different features for different classes by applying the information gain or gain ratio in order to identify relevant features for each binary classifier. Moreover, applying the information gain or gain ratio will return all the features that contain more information for separating the current class from all other classes. The output of this ensemble of binary classifiers will be decided using arbitration function based on the confidence level of the output of individual binary classifiers.

3.3. Dataset Aggregation Process:

It trains a deterministic policy that achieves good performance guarantees under its induced distribution of states. In its simplest form, the algorithm proceeds as follows. At the first iteration, it uses the expert's policy to gather a dataset of

trajectories D and train a policy that best mimics the expert on those trajectories. Then at iteration n, it uses ^n to collect more trajectories and adds those trajectories to the dataset D. The next policy is the policy that best mimics the expert on the whole dataset D.

3.3 Feature Selection or Dimensionality Reduction:

Feature selection or reduction keeps the original features as such and select subset of features that predicts the target class variable with maximum classification accuracy. It uses the intended learning to evaluate the usefulness of features, while filter evaluates features according to heuristics based on general characteristics of the data. The wrapper approach is generally considered to produce better feature subsets but runs much more slowly than a filter.

IV. CONCLUSION AND FUTURE WORK

In this work, we have achieved query cost minimization, performance, efficiency, etc., through DAGGER algorithm. DAGGER proceeds by collecting a dataset at each iteration under the current policy and trains the next policy under the aggregate of all collected datasets. The intuition behind this algorithm is that over the iterations, we are building up the set of inputs that the learned policy is likely to encounter during its execution based on previous experience (training iterations). This algorithm can be interpreted as a Follow-The-Leader algorithm in that at iteration n and pick the best policy in hindsight, i.e. under all trajectories seen so far over the iterations. The system then produces its best approximate result for that penalty function. Proposed techniques for online aggregation provide a way to simultaneously answer a batch of queries using progressive approximations of the underlying dataset. Priority can be placed on different cells, giving users control over a form of structural error. For appropriate data this technique provides accurate answers quickly, but the entire relation must be viewed before results become exact. This paper explores how query approximation can be used as an alternative to data approximation to provide efficient progressive query answering tuned to an arbitrary penalty function.

Provides a way to simultaneously answer a batch of queries using progressive approximations of the underlying dataset. Range aggregation a multi-bucket value source based aggregation that enables the user to define a set of ranges - each representing a bucket. During the aggregation process, the values extracted from each document will be checked against each bucket range and "bucket" the relevant/matching document. Bundled range aggregation, which can be regarded as the simultaneous execution of a range aggregate query on multiple datasets, returning a result for each dataset. B-trees are balanced trees that are optimized for situations when part or all of the tree must be maintained in secondary storage such as a magnetic disk. Since disk accesses are expensive (time consuming) operations, a b-tree tries to minimize the number of disk accesses. We are not aware of any previous study dedicated to bundled range aggregation. This problem, given its importance both as a standalone operator and as the building brick of more complex problems, deserves specialized efforts to improve the above straightforward methods. It present DAGGER (Dataset Aggregation), an iterative algorithm that trains a deterministic policy that achieves good performance guarantees under its induced distribution of states. In its simplest form, the algorithm proceeds as follows. The proposed method solves the greater than B page size problem. DAGGER (Dataset Aggregation), an iterative algorithm that trains a deterministic policy that achieves good performance guarantees under its induced distribution of states

REFERENCES

[1] Yufei Tao, Cheng Sheng, "I/O-Efficient Bundled Range Aggregation", IEEE Transactions on Knowledge & Data Engineering, , no. 1, pp. 1.

[2] D. Papadias, Y. Tao, P. Kalnis, and J. Zhang, “Indexing spatio- temporaldata warehouses,” in Proc. 18th ICDE, San Jose, CA,USA, 2002, pp. 166–175.

[3] R. R. Schmidt and C. Shahabi, “How to evaluate multiple range-sum queries progressively,” in Proc. PODS, 2002, pp. 133–141.

[4] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. E. Hambrusch, “Indexing uncertain categorical data,” in Proc. ICDE, 2007,pp. 616–625.

[5] Betty Salzberg , Vassilis J. Tsotras, “Comparison of access methods for time-evolving data", ACM Computing Surveys (CSUR), v.31 n.2, p.158-221, June 1999

[6] J. Yang and J. Widom, "Incremental computation and maintenance of temporal aggregates," VLDB J., vol. 12, no. 3,pp. 262–283, 2003.

[7] N. Sarkas, G. Das, N. Koudas, and A. K. H. Tung, "Categorical skylines for streaming data," in Proc. SIGMOD, Vancouver, BC, Canada, 2008, pp. 239–250.

[8] J. V. den Bercken, B. Seeger, and P. Widmayer, "A generic approach to bulk loading multidimensional index structures," in Proc. VLDB, Athens, Greece, 1997, pp. 406–415.

[9] S. Geffner, D. Agrawal, A. E. Abbadi, and T. R. Smith, "Relative prefix sums: An efficient approach for querying dynamic olap data cubes," inProc. ICDE, 1999, pp. 328–335.

[10] Bruno Becker , Stephan Gschwind , Thomas Ohler , Bernhard Seeger , Peter Widmayer," An asymptotically optimal multiversion B-tree". p.264-275.

[11] Bruce C. Huang, "Parallel Algorithms for Computing Temporal Aggregates, on Data Engineering" p.418, March 23-26, 1999

[12] Chee Yong Chan , Yannis E. Ioannidis, Hierarchical Prefix Cubes for Range-Sum Queries, Proceedings of the 25th International Conference on Very Large Data Bases, p.675-686, September 07-10, 1999

[13] Alok Aggarwal , Jeffrey,S. Vitter, The input/output complexity of sorting and related problems, Communications of the ACM, v.31 n.9, p.1116-1127.